REMARKS

Claims 1-3, 5, 6, 11, 16 and 17 have been amended to clarify the subject matter regarded as the invention. Claims 7-9 have been cancelled. New claims 21 and 22 have been added. Thus, claims 1-6 and 10-22 are now pending.

In the Office Action, the Examiner rejected claim 16 under 35 U.S.C. §112, second paragraph, as being indefinite for failing to particularly point out and distinctively claim the subject matter regarded as the invention. Claim 16 has been amended to depend on claim 11. Accordingly, it is respectfully requested that the Examiner withdraw this rejection.

In addition, the Examiner has rejected claims 1-7, 9, 11, 14-15, and 16-20 under 35 U.S.C. §103(a) as being unpatentable over the Admitted Prior Art in view of U.S. Patent No. 4,847,754 (*Obermarck*). This rejection is fully traversed below.

The present application pertains to a method for managing flow of messages between two software modules. As such, claim 1 recites determining whether a first message can be propagated to or from a first synchronization queue associated with one of the two software modules; and propagating the first message to or from the first synchronization queue when it is determined that the first and second messages can be propagated.

As noted in the specification, in the conventional models, access to the synchronization queue associated with a software module is limited to one thread at a time (Specification, page 3, lines 9-23). Accordingly, it is respectfully submitted that the Admitted Prior Art does not teach propagating a first message to or from a synchronization queue associated with a software module while a second message is propagated between the software modules. Claim 1, among other things, recites this feature. Thus, it is respectfully submitted that claim 1 is patentable over the Admitted Prior Art.

It is noted that *Obermarck* relates to a method for serializing process access to shared resources utilizing low-level atomic functions. It is further noted that *Obermarck* states that the task of performing a completion operation can be passed to a concurrently-accessing process. However, *Obermarck* does not teach concurrent propagating of messages to or from a synchronization queue in the context of the invention. In other words, *Obermarck* does not teach propagating a first message to or

from a first synchronization queue while allowing a second thread to propagate a second message between the software modules. As such, it is earnestly believed that *Obermarck* does not teach propagating a first message to or from a synchronization queue associated with a software module. Thus, it is respectfully submitted that claim 1 is patentable over *Obermarck* for at least this reason.

Accordingly, it is respectfully submitted that claim 1 is patentable over the Admitted Prior Art and *Obermarck* taken alone or in any proper combination. In addition, claims that are dependent on claim 1 are patentable at least for this reason. Moreover these dependent claims recite additional features which render them patentable for additional reasons.

Furthermore, independent claims 11 and 17 recite similar features as those recited in claim 1. For example, claim 11, among other things, recites a propagation controller operating to enable at least two processors of the plurality of processors to concurrently propagate messages to or from an auxiliary queue of a second software module. Accordingly, it is respectfully submitted that claims 11 and 17 and their dependent claims are patentable over the Admitted Prior Art and *Obermarck* taken alone or in any proper combination. Thus, it is respectfully requested that the Examiner withdraw all rejections under 35 U.S.C. §103(a).

SUMMARY

Based on the foregoing, it is submitted that claims 1-6 and 10-22 are patentably distinct over the cited art of record. Additional limitations recited in the independent claims or the dependent claims are not further discussed as the limitations discussed above are sufficient to distinguish the claimed invention from the cited art. Accordingly, it is respectfully requested that the Examiner withdraw all the rejections.

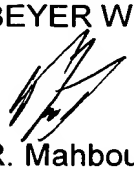
Applicant believes that all pending claims are allowable and respectfully requests a Notice of Allowance for this application from the Examiner. Should the Examiner believe that a telephone conference would expedite the prosecution of this application, the undersigned can be reached at the telephone number set out below.

If there are any issues remaining which the Examiner believes could be resolved through either a Supplemental Response or an Examiner's Amendment, the Examiner

is respectfully requested to contact the undersigned attorney at the telephone number listed below.

Applicants hereby petition for an extension of time which may be required to maintain the pendency of this case, and any required fee for such extension or any further fee required in connection with the filing of this Amendment is to be charged to Deposit Account No. 500388 (Order No. SUN1P398).

Respectfully submitted,
BEYER WEAVER & THOMAS, LLP



R. Mahboubian
Reg. No. 44,890

P.O. Box 778
Berkeley, CA 94704-0778
(650) 961-8300

1. (Once Amended) A method for managing flow of messages between two software modules, said method comprising:

(a) determining whether a first message can be propagated to or from a first synchronization queue associated with one of the two software modules, by a first thread running on a first processor, [between the two software modules] while allowing a second thread running on a second processor to propagate a second message between the two software modules; and

(b) propagating the first message [between] to or from the [two] first synchronization queue [software modules] while allowing the second thread to propagate the second message between the two software modules when said determining (a) determines that the first message can be propagated by the first thread while allowing the second thread to propagate the second message between the two software modules.

2. (Once Amended) A method as recited in claim 1, wherein the first and second threads concurrently propagate respective portions of the first and second messages [between] to or from the first synchronization queue [the two software modules].

3. (Once Amended) A method as recited in claim 1, wherein the method further comprises:

(c) blocking the second thread from propagating the second message between the two software modules when said determining (a) determines that the first message cannot be propagated by the first thread [while allowing the second message to propagate between the two layer software modules].

5. (Once Amended) A method as recited in claim 1, wherein the method further comprises:

setting a first indicator for the first processor to indicate that the first processor is propagating when said determining (a) determines that the first message can be propagated by the first thread while allowing the second thread to propagate the second message between the two [layer] software modules[.] ;and

setting the first indicator for the first processor to indicate that the first processor is not propagating when said propagating (b) has propagated the first message between the two software modules.

6. (Once Amended) A method as recited in claim 1, wherein said determining (a) comprises:

(a1) determining whether an event is being processed or is pending to be processed; and

(a2) determining whether a thread-count for the first processor is zero, and

wherein said determining (a) determines that the first thread can propagate the first message without blocking the second thread from propagating the second message when said determining (a) determines that no events is being processed or pending and [said determining (a) determines that] the thread-count for the first processor is zero.

7. (Cancelled)

8. (Cancelled)

9. (Cancelled)

11. (Once Amended) A computer system comprising:

a plurality of processors;

first and second software modules, the second software module having a main queue suitable for storing messages and an auxiliary queue suitable for storing messages that are not stored in the main queue; and

a propagation controller for propagating messages between the first and the second software modules, the propagation controller operating to enable at least two processors of said plurality of processors to concurrently propagate messages [between the first and the second software modules] to or from the auxiliary queue of the second software module.

16. (Once Amended) A computer system as recited in claim [16] 11, wherein the first and second software modules are implemented in a stack as STREAMS modules.

17. (Once Amended) A computer readable media including computer program code for managing flow of messages between two software modules, said computer readable media comprising:

computer program code for determining whether a first message can be propagated by a first thread running on a first processor [between] to or from a first

synchronization queue of one of the two software modules while allowing a second thread running on a second processor to propagate a second message between the two software modules; and

computer program code for propagating the first message to or from the first synchronization queue [between the two software modules] while allowing the second thread to propagate the second message between the two software modules when said computer program code for determining determines that the first message can be propagated by the first thread while allowing the second thread to propagate the second message between the two software modules.

18. (Once Amended) A computer readable media as recited in claim 17, wherein the first and second threads concurrently propagate respective portions of the first and second messages to or from the first synchronization queue [between the two software modules].

APPENDIX A

1. (Once Amended) A method for managing flow of messages between two software modules, said method comprising:

(a) determining whether a first message can be propagated to or from a first synchronization queue associated with one of the two software modules, by a first thread running on a first processor, while allowing a second thread running on a second processor to propagate a second message between the two software modules; and

(b) propagating the first message to or from the first synchronization queue while allowing the second thread to propagate the second message between the two software modules when said determining (a) determines that the first message can be propagated by the first thread while allowing the second thread to propagate the second message between the two software modules.

2. (Once Amended) A method as recited in claim 1, wherein the first and second threads concurrently propagate respective portions of the first and second messages to or from the first synchronization queue.

3. (Once Amended) A method as recited in claim 1, wherein the method further comprises:

(c) blocking the second thread from propagating the second message between the two software modules when said determining (a) determines that the first message cannot be propagated by the first thread.

4. A method as recited in claim 2, wherein said blocking (c) of the second thread from propagating the message between the two software modules is achieved by providing a lock which can be acquired by the first thread.

5. (Once Amended) A method as recited in claim 1, wherein the method further comprises:

setting a first indicator for the first processor to indicate that the first processor is propagating when said determining (a) determines that the first message can be propagated by the first thread while allowing the second thread to propagate the second message between the two software modules; and

setting the first indicator for the first processor to indicate that the first processor is not propagating when said propagating (b) has propagated the first message between the two software modules.

6. (Once Amended) A method as recited in claim 1, wherein said determining (a) comprises:

(a1) determining whether an event is being processed or is pending to be processed; and

(a2) determining whether a thread-count for the first processor is zero, and

wherein said determining (a) determines that the first thread can propagate the first message without blocking the second thread from propagating the second message when said determining (a) determines that no events is being processed or pending and the thread-count for the first processor is zero.

7. (Cancelled)

8. (Cancelled)

9. (Cancelled)

10. A method as recited in claim 1, wherein the two software modules are implemented in a stack as STREAMS modules.

11. (Once Amended) A computer system comprising:

a plurality of processors;

first and second software modules, the second software module having a main queue suitable for storing messages and an auxiliary queue suitable for storing messages that are not stored in the main queue; and

a propagation controller for propagating messages between the first and the second software modules, the propagation controller operating to enable at least two processors of said plurality of processors to concurrently propagate messages to or from the auxiliary queue of the second software module.

12. A computer system as recited in claim 11,

wherein the propagation controller comprises:

a thread-count for one of said plurality of processors; and

a queue count for the auxiliary queue.

13. A computer system as recited in claim 12,

wherein the auxiliary queue is a synchronization queue and the queue count is a synchronization queue count.

14. A computer system as recited in claim 11, wherein the at least two processors of said plurality of processors concurrently propagate a message from the first software module to the auxiliary queue of the second software module.

15. A computer system as recited in claim 11, wherein the at least two processors of said plurality of processors concurrently propagate a message from the first software module to the main queue of the second software module.

16. (Once Amended) A computer system as recited in claim 11, wherein the first and second software modules are implemented in a stack as STREAMS modules.

17. (Once Amended) A computer readable media including computer program code for managing flow of messages between two software modules, said computer readable media comprising:

computer program code for determining whether a first message can be propagated by a first thread running on a first processor to or from a first synchronization queue of one of the two software modules while allowing a second thread running on a second processor to propagate a second message between the two software modules; and

computer program code for propagating the first message to or from the first synchronization queue while allowing the second thread to propagate the second message between the two software modules when said computer program code for determining determines that the first message can be propagated by the first thread while allowing the second thread to propagate the second message between the two software modules.

18. (Once Amended) A computer readable media as recited in claim 17, wherein the first and second threads concurrently propagate respective portions of the first and second messages to or from the first synchronization queue.

19. A computer readable media as recited in claim 18, wherein the computer readable media further comprises:

computer program code for setting a first indicator for the first processor to indicate that the first processor is propagating when said determining (a) determines that the first message can be propagated by the first thread while allowing the second thread to propagate the second message between the two layer software modules.

computer program code for setting the first indicator for the first processor to indicate that the first processor is not propagating when said propagating (b) has propagated the first message between the two software modules.

20. A computer readable media as recited in claim 19, wherein the two software modules are implemented in a stack as STREAMS modules.

21. (New) A method as recited in claim 1, wherein the second thread also propagates the second message to or form the first synchronization queue.

22. (New) A method as recited in claim 1, wherein the second thread propagates the second message to or form one of the first synchronization queue and a second synchronization queue, the second synchronization queue being associated with one of the software modules.